

# On-line Text Editing: A Survey

ANDRIES VAN DAM AND DAVID E. RICE

*Center for Computer and Information Sciences, Brown University, Providence, Rhode Island*

This paper is a survey of current methods for the on-line creation and editing of computer programs and of ordinary manuscript text. The characteristics of on-line editing systems are examined and examples of various implementations are described in three categories: program editors, text editors, and terminals with local editing facilities.

*Key words and phrases:* on-line editing, program editing, text editing, editing terminals, command languages

*CR categories:* 1.3, 3.9, 4.30, 6.35

## INTRODUCTION

With the advent of inexpensive terminals that communicate directly with a general-purpose computer, there has been a noticeable movement in the computing industry towards utilizing the resources of the computer in many new, non-numerical applications. For example, on-line creation and modification of programs and their documentation have become widely accepted as productive and cost-effective uses of the computer. In fact, it has been realized that the facilities provided by a time-sharing system's central editing program and its command language are among the most important determinants of the system's convenience, power, and consequent utility. Along the same lines, special-purpose computer-assisted text editing packages have become accepted in industry and government for the preparation or printing of technical manuals, proposals, and other documents in which many updates are necessary and revision time is at a premium. While these packages are primarily *hard-copy* output oriented, exciting research has been done, notably at the Stanford Research Institute, on the ways in which display console (soft-copy) oriented text- (and picture-) handling systems can enhance all

aspects of the user's programming, writing, and thinking.

The purpose of this article is to survey various representative types of text manipulation systems from the user's point of view, allowing him to see what facilities are available today. Implementation details of on-line systems, particularly with respect to data structure manipulation, are covered in [17]. First, a general overview of a typical system is presented. Distinguishing characteristics of a representative sample of systems are then covered, using a (somewhat artificial) partitioning of these systems into the two classes of "program editors" and "free-form text editors." The hardware/firmware editing features of typical display consoles are also briefly discussed.

### Typical Structure of a Text Editor

The simplified functional block diagram in Figure 1 is representative of most editors, though the overall system in a particular case need not be segmented into the exact set of subprograms shown. The user is seated at a typewriter terminal or a cathode-ray tube display console (CRT) on which one or more lines of characters of his file are visible. The terminal in effect provides the user with a "window" on his file. The user's commands to alter text displayed

**CONTENTS**

|                                                   |         |
|---------------------------------------------------|---------|
| Introduction                                      | 93-97   |
| Typical Structure of a Text Editor                |         |
| Distinguishing Characteristics                    |         |
| Program Editors                                   | 97-105  |
| Conversational Context-Directed Editor, IBM       |         |
| Cambridge Scientific Center                       |         |
| WYLBUR, Stanford University                       |         |
| Quick EDitor (QED), Com-Share                     |         |
| Text Editor and COrrector (TECO), MIT             |         |
| TVEDIT, Stanford University                       |         |
| Interactive Programming Support System, System    |         |
| Development Corporation                           |         |
| Emily, Argonne National Laboratory                |         |
| Free Form Text Editors                            | 105-111 |
| Magnetic Tape Selectric Typewriter (MTST), IBM    |         |
| Corporation, and ASTROTYPE, Information Control   |         |
| Systems                                           |         |
| System/360 Administrative Terminal System, IBM    |         |
| Corporation, and VIPcom, VIP Systems              |         |
| A Tablet-Based Editor, Carnegie-Mellon University |         |
| The Hypertext Editing System (HES) and File       |         |
| Retrieval and Editing SyStem (FRESS), Brown       |         |
| University                                        |         |
| NLS, Augmented Human Intellect Research Center,   |         |
| Stanford Research Institute                       |         |
| Editing Features of Typical Terminals             | 111-113 |
| Conclusion                                        | 113     |
| Postscript                                        | 113-114 |
| References                                        | 114     |

in the window are entered via the typewriter keyboard (or function keys, lightpen, or other input devices), while the physical/logical attention (interrupt) generated when he transmits a command is fielded by the Input Handler to a Command Interpreter. This routine parses the requested command, typically into an editing operation (insert, delete, substitute, rearrange, etc.) and associated data. The latter consists primarily of positioning information (e.g., at what point in the text an insertion is to be made, or between what two points in the text a deletion is to be made) and literal character string data (e.g., the text to be inserted). In addition to editing, the user may "travel" through the text in order either to read or to make changes in a different section. The command in this case might be to advance the display window in the file, and the associated data field would indicate the number of display lines to be advanced ("scrolled") and the direction (forwards or backwards).

The parsed (internal) form of the request is passed to an appropriate editing or traveling routine that performs the actual operation. The internal form of the text (alternatively called the data or storage structure\*) is altered (in the case of an edit), and the updated text is reformatted by the Display Generator for feedback to the user. When the files are large, and traveling or large edits are invoked, the relevant portion of the storage structure may not be resident in core, in which case the Paging Routines must be called upon to replace some inactive data in core with the data requested from secondary storage [6].

In the general case, the system supports multiple terminals, and a time-sharing monitor/file-handler is superimposed on the block diagram. It has basic responsibility for supervising the sharing of programs and core among users while guaranteeing the integrity of their files. It may also provide other services, such as user-to-user messaging or even file sharing for cooperative efforts.

\* More precisely, the machine-independent structure of the file may be referred to as the data structure, while its machine-dependent encoding would be the storage structure.



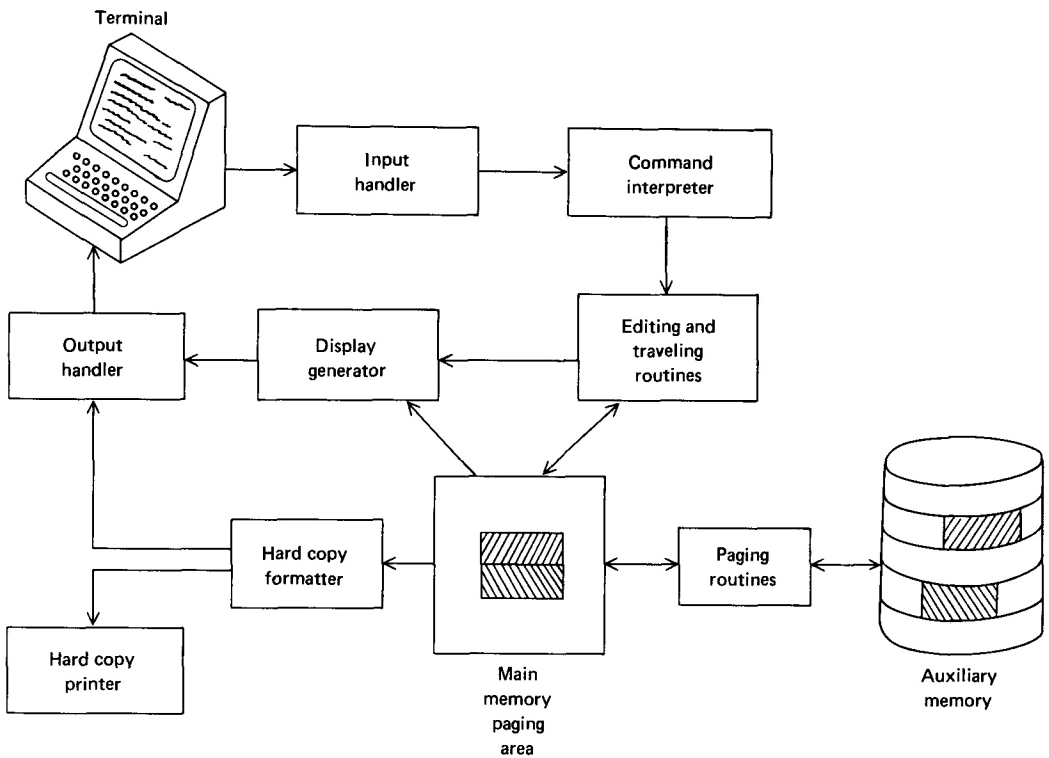


FIG. 1. Overview of an on-line editor.

The Hard-Copy Formatter (which may in fact be largely merged with the Display Generator) is used in the case of free-form text (as opposed to program text) to convert the storage structure of the file to conventional hard copy for output on typewriter terminals, highspeed line printers, or even photo-composition devices. In his text, the user may specify "format codes" (typesetting codes) that determine margins, headings, running heads, paragraphs, left and/or right margin justification, indents, centering, underscores, type-face changes, etc. These codes are frequently stored in-line with the text itself and may be treated as indistinguishable from text for editing purposes. Sophisticated formatters may produce other byproducts useful for hard copy: foot-notes, hyphenation, tables of contents, lists of figures, indexes, spelling checks, etc. Very readable accounts of the recent advances in computer-assisted typesetting and printing are contained in [11] and [25].

The generic design goals of on-line editors, from the user's point of view, are:

- 1) fast response to a large number of terminals (which ideally are designed with good "human factors" for the editing task);
  - 2) a concise, mnemonic command language with informative feedback;
  - 3) powerful commands, with few restrictions and exceptions, to make possible anything that one can do to hard copy with pencil, scissors, and staples, or anything that one can do to program card decks; and
  - 4) commands that take advantage of computer capabilities; e.g., traveling to the first occurrence of a user-specified pattern in the file (called pattern, content, or context scanning); "uniform" substitution of one pattern for another every place it occurs, or only in the first instance, or only if the pattern occurs in a certain column-dependent field; automatic renumbering of sections or references after the file is altered; and flexible hard-copy output.
- Naturally, all of these goals are to be met

using as little as possible of such resources as money, implementation time, CPU cycles, core and disk space, etc. Furthermore, the goals are both interdependent and conflicting since, for example, increased power in the instruction repertoire implies greater demand on resources and therefore slower response time (or support of fewer terminals).

### **Distinguishing Characteristics**

The storage method (the data/storage structure employed) and the editing functions provided by a given system are strongly influenced by the type of text upon which the system is intended to operate: computer programs are typically manipulated with "program editors" while ordinary manuscripts are edited with "text editors." On the other hand, the manner in which the editing functions are actually specified and the way text is displayed depend primarily on the type of terminal used, either a teletype/typewriter terminal or a CRT display.

The first of the distinguishing characteristics between editors, then, is in the type of editing to be performed and the attendant structure of the file. When editing a program, one typically modifies the program "in place," substituting one small string of text, such as an op-code or an operand address, for another, or inserting a label in a field (a portion of the line) that was previously blank. In such cases it is quite reasonable to store the text line by line (80 or 132 columns/line) in card image or printer line image format. With an editor oriented towards free-form text, on the other hand, one wants to make insertions and substitutions of arbitrarily sized character strings at arbitrary points in the manuscript. This increased flexibility implies that the data/storage structure should be more flexible to cope with the more powerful types of insertions and deletions (e.g., overflowing or contracting automatically from line to line, within a paragraph). Thus, one typically finds that the unit of storage within which text may grow or shrink dynamically is a "super" line or statement of several hundred characters, a paragraph, or even an entire "page."

In both program and text editors, it is possible to view the file as more than a collection of lines or literal character strings. For example, it is useful to be able to present the block structure of a procedural-language program or the outline hierarchy of a section-numbered text by suitable formatting (line skips and indentation) on a display screen (see the sections on Emily and NLS). Naturally, this block structure must be encoded in the data structure.

While it is entirely possible (and in fact, quite common) to use a given editor for either text or program editing, the design objectives and the resultant capabilities are usually specialized and more convenient for one than the other. (The NLS system [9] in which the editing system and language processors were designed together to form an integrated system is a striking exception.) A rough classification of existing editors along these lines of specialization is therefore possible.

A second distinction is in the method of specifying operations; this is primarily based on the type of terminal employed. Any editing task requires two inputs to the computer: the task to be done and identification of the portion of text to which it applies. If a teletypewriter is the input device, the task to be done is supplied by typing the name (or an abbreviation) of the command. The location in the text to which the command applies is identified by giving a line number, by typing an appropriate context string, or both. In most systems using them, line numbers are defined as the displacements of lines relative to the top of the file; they are therefore rather arbitrary since they can change dynamically as the file is changed, and in any case they bear little relationship to the actual text. With context identification, a user-specified string of characters (text) is searched for in the text to locate the desired position. Having to specify context often requires extra work for the user since he must type enough characters to identify the location uniquely; otherwise unintentional ambiguities might be introduced. However, the character strings in the file are not subject to change by the system (as relative line numbers are) and consequently context identi-

cation typically provides a more stable method of locating portions of a file.

When a CRT display unit is used with some mechanism for pointing at the desired location, two improvements in location identification are made: text is rapidly displayed and identification can be made by "pointing." The pointing can be done either directly, as with a lightpen or a data-tablet stylus, or indirectly, by overlaying the requisite character with a pointer symbol or "cursor" driven by keys (up, down, left, right), a "joystick," or a "mouse" (a hand-held X-Y transducer usable on any flat surface). Thus, teletypewriters require the user to locate the text in question explicitly, while CRT displays usually allow the user to point, relegating to the editing system the task of correlating the indicated point on the screen with the corresponding position in the internal data structure (typically via "a correlation map").

A third distinction pertains to the manner in which the text is presented. Typewriter-like devices must print (display) text mechanically and are therefore excruciatingly slow (approximately 100–300 words/minute). As a result, the amount of text that may be displayed at one time for editing purposes is quite small (usually only one line). The tremendous advantage in using a CRT display is that the text is "printed" at electronic speeds, thus allowing many lines to be displayed at each stage of the editing process. Typewriter-like devices are so slow that they force the user to work from a mocked-up printed copy, manually transcribing changes already made in the printed copy, and therefore doing the work twice. With a CRT display, the editor may think out and implement his changes simultaneously. Naturally, teletypewriter-equivalent CRTs that are connected over the same bandwidth line as a teletypewriter offer only the advantage of silent operation, at the disadvantage of not producing a hard-copy transcript of the session.

The fourth distinction is in the functions for producing output and the form of the output. In a program editor, the text is displayed and printed as it is stored, line by line. The only output facilities required are a "new line" function and possibly a tab-

bing facility; only upper case characters need be used. In contrast, most text editors consider the displayed line and the printed line as two distinct, temporary units to be derived from the internal data structure. The displayed lines may be simple mappings of the internal form, showing, for example, only skipped lines and paragraphs, while the printed version may be suitably structured by arbitrarily complex typesetting codes.

## PROGRAM EDITORS

Computer program editors have become increasingly popular in time-sharing environments, where a facility for the on-line modification of programs combined with some form of remote job entry (execution) greatly increases a programmer's productivity. Immediate verification of correct syntax of the program is occasionally provided.

The most common method for editing a program in a non-time-sharing environment is still to insert and delete cards from a punched source deck, which then has to be read back into the computer through the card reader. The dropping or loss of decks evokes bitter memories for all programmers. Also, backspacing over typographical errors with a keypunch is not recommended. An on-line editing system that stores the programs on disk or tape eliminates the need for hand-carrying card decks to and from the machine, and enables updates to be made quickly and efficiently. When used in conjunction with a time-sharing system under which the source program can be readily compiled, the time required to write and debug a program is greatly reduced.

Since many users have (potential) access to program editors, it was felt by the authors that actual examples of the use of various system features would be desirable; simple transcripts are therefore provided below for a few of the most common editors.

### **Conversational Context-Directed Editor, IBM Cambridge Scientific Center**

A fairly typical program editor is the Conversational Context-Directed Editor

developed at the IBM Cambridge Scientific Center for the 360/67 CP/CMS operating system and known more widely as the CMS editor [5]. Its facilities, like those of most program editors, may be traced to the trendsetting context editor developed in the early 1960s for project MAC's CTSS time-sharing system for the IBM 7090. Although the CMS editor is often used to edit normal text, the editing commands and the interactive terminals (IBM 2741s) are best suited for the simplified text of a computer program.

The way that the text is stored internally depends on the type of text being edited. The text of computer programs is divided into fixed-length (80-character) records, corresponding one-for-one with a standard card deck of source code. This one-for-one storage scheme is the easiest to implement but is certainly not the most compact form of storage because most of a record (card) is normally left blank. Files of normal text (filetype SCRIPT) are stored as variable-length records (one line/record, with a maximum of 130 characters per record), thus providing for a substantial savings in the disk space required to store a large on-line data base. When the text is in main memory, however, the lines are treated as fixed in length and padded to 130 characters with blanks.

Since the editor was written for a computer with virtual memory, the entire file currently being edited is kept core-resident, with individual lines connected in a two-way linked list structure for ease of traveling and multi-line editing. The primary advantage of the virtual memory technique is that no programmed I/O to secondary storage is done by the editing program while the user is editing a file, and as a result response times are normally quite good. Of course, hardware paging to auxiliary memory is done by the operating system [6]. The risk in keeping an entire file core-resident is that if the machine or operating system should fail as the user is editing, all his work since he last saved the current file on disk is lost. In addition, the memory size of the virtual machine determines the maximum size of a file.

The CMS editor provides two modes for program modification. In the "input" mode,

new text is continually entered, one line at a time. This mode is used to create a program or to insert several contiguous lines of code into an existing program. In the "edit" mode, text can be inserted by line, one or more lines may be deleted, and a single line may be retyped. Changes within a line are made by typing the incorrect characters followed by the replacement string; inserts are made by replacing the character string immediately in front of the insert point by a string consisting of this context and the string to be inserted, while deletes are done by a null replacement for the string to be deleted. This method for specifying editing changes is referred to as *context-directed* editing. For example, to change ORANGES to GRAPEFRUITS in the line:

SUM = APPLES + ORANGES

the user would type:

change /oranges/grapefruits/

As the previous example indicates, replacements within a line need not be of the same length; however, if the line length exceeds the maximum length for the particular type of file, the extra characters are truncated. Thus a large insertion in the middle of a line must be broken into separate lines to avoid this overflow problem. This restriction is not serious for program editing but is often very distracting for general-purpose editing. Universal replacements of one phrase for another within a file (such as changing a label and all of its references) can be done quite easily with the "change" command.

Since the CMS editor is line-oriented, the current line in the program is determined by a "line pointer" that changes as traveling and editing occur. The current line is defined as that line being created or edited in the file at a given moment. The line pointer may be moved backwards or forwards by one or more lines (following the backwards or forwards pointer chain) to accomplish scrolling through the file. There are also two useful commands that search for a specified character string occurring either in a fixed line position ("find") or anywhere in a line ("locate"). If a match is found, the line pointer is moved to the line

located. Hence, if a character string that the user specifies while looking at formatted (justified) hard copy happens to be spread over two internal lines, it will not be found. Obviously, this situation only arises for text, and not for program editing.

As an example of how a line-oriented program editor might be used, assume that the following section of a program, which computes the sum of two matrices, is to be modified to compute the difference of the two matrices:

```
ADD: FOR ROW = 1 TO N DO;
      FOR COLUMN = 1 TO M DO;
        C (ROW, COLUMN) = A (ROW, COLUMN) + B (ROW, COLUMN);
      END;
END;
```

The following sequence of interactions with the editor would provide the necessary changes (the user's requests are in lower case and the machine responses are in capitals):

```
find add
ADD: FOR ROW = 1 TO N DO;
change /add/subtract/
SUBTRACT: FOR ROW = 1 TO N DO;
next 2
C(ROW, COLUMN) = A (ROW, COLUMN) + B (ROW, COLUMN)
change /+/-/
C (ROW, COLUMN) = A (ROW, COLUMN) - B (ROW, COLUMN)
top
change /add/subtract/ * *
```

The routine ADD is first located by using the "find" command, which searches for the string ADD beginning in the first position of the line. Next, the label ADD is changed to SUBTRACT. (The slashes (/) are used as delimiters and can be any character not occurring in either the old or new strings.) Next, the line pointer is moved ahead two lines by the command "next 2" to locate the next line to be changed; then the addition (+) is changed to subtraction (-). The remaining two commands search the entire program (starting at the "top") to uniformly replace all occurrences of ADD with SUBTRACT, so that any references to the ADD routine now refer to SUBTRACT.

The CMS editor also provides flexible tabbing facilities that require less typing for the user when column-dependent languages such as FORTRAN are being used. A tabbing facility also makes the programmer more likely to indicate block structures and nesting levels by suitable indentations, a useful documentation technique. Default

(internal) tab settings appropriate to the specific language are automatically applied to the file. For example, if the program were being written in FORTRAN, the tab stops would be at positions 7, 10, 15, 20, 25, and 30.

### WYLBUR, Stanford University

WYLBUR is a line-oriented text editor developed by the Stanford Computation Center for use on 2741 terminals to provide an on-line editing facility that is used in conjunction with the on-line and batch services of Stanford's IBM 360/67 [26].

A WYLBUR file is divided into lines of text, each containing from 0 to 133 characters; but normally a maximum of 72 characters is used for programs. Every line is stored as an unpadding character string with an associated length and line number. This seems to be a more reasonable data structure for the storage of computer programs, which usually leave most of a line (card) blank. Text is still line addressable, but the storage is more compact. The line numbers are absolute (as opposed to relative to the top of the file) and do not change dynamically as editing is performed; this provides stable reference points.

WYLBUR editing commands generally operate on a *range* of lines. A range is defined as a group of lines that all possess the same distinguishing characteristic. Ranges can be specified in several ways. An *explicit* range can be either all lines whose line numbers fall within a given range, such as lines 3 through 12 (indicated as 3/12), or simply a list of line numbers, such as lines 7, 12, and 16.5 (indicated as 7, 12, 16.5). Both types of explicit ranges may be combined so that "5, 9.1, 13.02/14.5, 20" would indicate lines 5, 9.1, 20, and 13.02 through 14.5. The second type of range is the *associative* range, which includes all lines containing a given string of characters, e.g., all lines containing the string "COMPUTER." Explicit and associative ranges can also be combined: "COMPUTER" IN 6/12 would indicate the range of all lines between lines 6 and 12 containing the string COMPUTER.

The interline editing commands DELETE, MOVE, and COPY all operate on ranges of lines so that, for example, all lines

between 6 and 12 containing the string COMPUTER can be deleted with one editing command.

Intraline editing is done with the MODIFY command. The line to be modified is typed at the terminal, and the user indicates changes by typing special characters under the typed line. For example, he can delete part of the line by typing a "D" under the first and last characters of the string to be deleted; he can insert a string into a line by typing an "I" under the character before which the insert is to go, followed by the character string to be inserted (total length not to exceed the maximum line length of the file); and he can replace a same-length string by typing an "R" followed by the replacement characters. Replacing strings of different lengths can be done using the "D" and "I" forms.

Uniform changes can also be made to a range of lines using the CHANGE command; for example:

CHANGE "ADD" TO "SUBTRACT" IN ALL  
changes the string ADD to SUBTRACT everywhere it occurs in the file. The keyword ALL specifies the range of all lines in the file.

In addition to the editing facilities offered, WYLBUR provides a complete group of commands for sending programs to the batch processor to be assembled or compiled, and for listing the current status of a job in progress.

#### Quick Editor (QED), Com-Share

One of the best known examples of a teletype-based program editor utilizing variable length lines is the Quick Editor (QED) developed originally by the University of California at Berkeley and revised extensively for commercial use by Com-Share [7, 15]. It was designed expressly to provide maximum convenience for the user via a simple and mnemonic command language and line-independent access to the text.

QED is internally superline oriented. However, text is not stored as a fixed length record or a variable length record with an associated length. Instead, the end of the line is delimited by an internal marker. A line can be up to 500 characters long, although

only 80 are printed per teletype-line (i.e., it prints 80 character lines until it has exhausted the 500 characters).

QED has provisions for content addressing similar to those of the CMS Editor, providing a natural means of locating sections of text. QED can search for a preassigned label beginning in the first position of a line, or for an arbitrary string anywhere in the text.

The editing commands include insert (one or more complete lines), delete (one or more complete lines), change (i.e., replace one or more complete lines), and substitute (replace only part of a line; this is not restricted to one-for-one character replacements but the replacement must be less than the teletype line). In addition, there are control characters that are used to make more complex (but still minor) editing changes within a line.

An example of a simple QED editing sequence is:

```
*7INSERT (CR)
100 DO 101 I=1,N (CR)
101 SUM(I)=A(I) + B(I) (DC)
```

This sequence inputs two lines into a FORTRAN program preceding the 7th line in the buffer. The asterisk is typed by QED to indicate its readiness for a new command. Lines are ended with the carriage return symbol (CR), and the insert is ended with the special symbol (DC). A more powerful example is:

```
*4,10;/ADD/SUBTRACT/(CR)
```

which changes the string ADD to SUBTRACT each time it is found in lines 4 through 10.

An interesting feature of QED is that a sequence of editing operations can be saved by the system as a normal text file. At a later time, this set of commands can be re-executed (after possible modification of the commands) to re-edit the text. Of course, if the original text file has since been modified, the editing commands may no longer be defined. The re-editing facility makes it possible to maintain slightly different versions of a file without having to duplicate the main

file many times. The text is stored only once; the alternate versions are described by additional short files that contain only the appropriate editing changes. This is useful for testing changes to an operational program without actually modifying the original program until the changes have been checked out.

#### Text Editor and Corrector (TECO), MIT

Another well known program editor for teletypes is TECO, developed by the Massachusetts Institute of Technology and Project MAC for Digital Equipment Corporation's PDP series of computers [20]. It is commercially available from Interactive Sciences Corporation on the PDP-10.

TECO editing commands are more primitive than QED's; however, the fundamental set of commands is used as a set of building blocks that may be combined to provide quite elaborate editing operations. The drawback is that the user must think on a level lower than that required by other systems, and frequently must construct a little "program" of the basic commands to perform the edit. Among the fundamental commands are insert, delete, context scanning, and conditional execution.

Unlike the previous editors, TECO is not line oriented. A file is a sequence of characters, normally broken into variable length "pages" by Form Feed characters (which cause page ejection in hard-copy printout). Pages may be combined in an in-core "buffer," but normally one page at a time is edited in this buffer. After a buffer has been edited, it is written sequentially onto a second file, and the next page is accessed from the original file (or a different one). Because of this sequential organization, backing up to a previous page is not permitted; one must re-read the newly created file from the beginning.

The current position in the in-core buffer (displacement into the buffer) is determined by a character pointer that points between characters. This pointer can be positioned absolutely (by a numeric value), relatively (by a positive or negative character or line displacement), or by pattern searches. "Q registers" are available both for doing arithmetic and for moving or

copying strings of text. If a Q register contains text, the text may be interpreted as a command string (a macro facility). To move a string of text, for example, the string would first be saved in a Q-register, and then deleted from the buffer. Next, the character pointer would be moved to a new location and the contents of the Q-register copied into the buffer at this new point.

As an example of "bootstrapped" commands, consider the schematic diagram of a program that substitutes "string2" for "string1" uniformly in a file (see Figure 2). Whenever a context pattern ("string1") is found, the character pointer (called P in Figure 2) is automatically positioned after

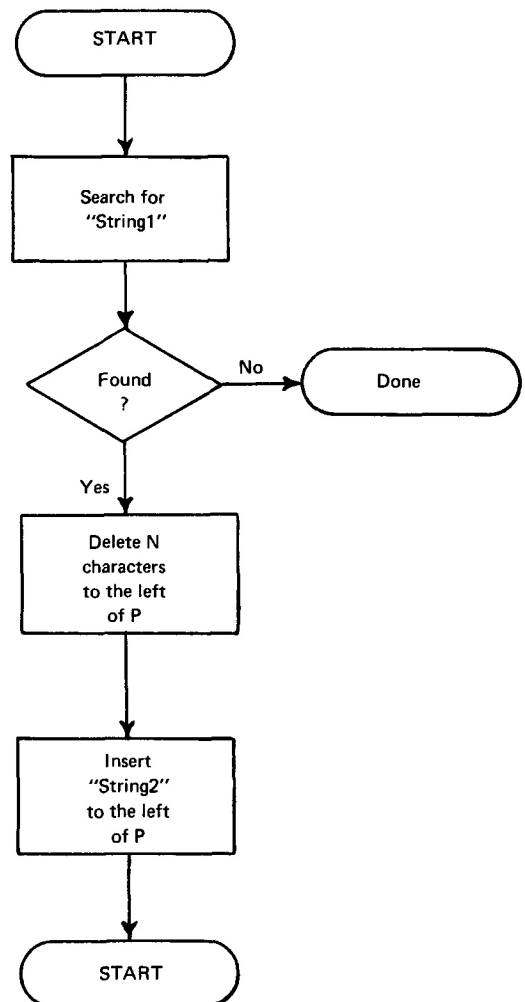


FIG. 2. Schematic representation of a universal substitute in TECO.

the last character of the pattern. The length of "string1" in Figure 2 is "N."

To uniformly substitute "grapefruits" for "oranges" wherever "oranges" occurs, one specifies the program:

```
J<SORANGES$;-7DIGRAPEFRUIT$>
```

"J" denotes jump character pointer to the left of the first character in the buffer; "<" and ">" denote the scope of commands in the iteration; "S" denotes search for the pattern that immediately follows (ORANGES) and is ended by the "\$," positioning the character pointer to the right of the last character in the string (i.e., to the right of the matched "S"); ";" delimits the commands to be skipped in case of a scan failure, so that the next command to be executed is the one following the matching ">"; "-7D" denotes the deletion of the 7 characters to the left of the pointer, again leaving the pointer to the left of the character immediately following the "S" of the deleted "ORANGES"; "I" denotes insertion to the left of the pointer of the string immediately following (GRAPEFRUIT\$), up to the "\$."

#### **TVEDIT, Stanford University**

TVEDIT is one of the earliest (1965) time-sharing, CRT-based text editors [12, 22]; it displays many lines of text at electronic speed. The user thus continually views the most recent version of his text and does not have to constantly refer to a mocked-up copy. Users ordinarily create and maintain text without any hard copy, not even an initial hand-written draft.

Because of careful user engineering, the simple TVEDIT command language is almost invisible to the experienced user. A control shift button alters the interpretation of alphanumeric keys to perform required functions. A request for these functions can be preceded by a number to repeat the command that number of times. Commands are executed as they are typed, so the user never has to press an execute command key and is seldom forced to wait for a system response.

Line-oriented editing is performed by using a pointer displayed on the screen. The

pointer can point at either a line or between two characters and can be moved by typing space, backspace, and carriage return with the control shift down. Control/carriage return moves the pointer to the next line. Control/space moves the pointer to the next character or to the first character in the line if the pointer pointed at a line. Control/backspace moves the pointer back one character or line depending on which of these is indicated by the pointer. Note that these key combinations simulate the cursor control keys on alphanumeric editing consoles discussed in the section on editing features of typical terminals.

Characters can be replaced by simply typing the replacement over the old characters. Lines or characters are deleted by moving the pointer to just before the item and typing a control character. When deleting single characters, characters to the right of the deletion on the line move to the left so the pointer appears to "swallow" characters. Similarly, for insertions, the characters to the right of the pointer move to the right. Since only 50 characters can be displayed per line and lines can be up to 128 characters, lines longer than fifty characters will appear to be truncated on the display. A symbol appears to indicate this truncation.

To facilitate travel, the user may segment the text into "pages" by inserting page marks between any two lines of text. The travel commands are: go to the next page, go to the *n*th page, and display the next screenful of text. The system can display text almost as fast as the user can type the one-key command. As a result, users usually find text by requesting the approximate page and visually scanning forward to the desired view. No explicit pattern scanning commands are provided by the editor.

#### **Interactive Programming Support System, System Development Corporation**

This editor [2] has been designed for the IBM 2260 alphanumeric CRT display unit, making maximum use of its limited capabilities. The 2260 has a cursor (a "typing pointer") that may be driven around the screen, but does not have a user-held pointing device. As currently implemented, the



editor is used to create and edit programs written in a procedure-oriented programming language (JOVIAL). Each line, as it is entered, is syntax checked, a feature which is representative of a number of time-sharing system program editors.

The editing repertoire includes insert, delete, replace, copy, and move, all applying to one or more lines (but not within lines). The lines are numbered and same-length substitutions within a line can be made by specifying the line number, driving the cursor to the desired point with the space bar, and typing the correction in place. The line number is needed because there is no extensive correlation map between the data structure and the screen display. A continuous input mode (the "compose" mode) also exists, in which consecutive lines may be inserted without having to specify the insert command for each. The program tries to minimize user typing and, since it has been designed for a specific language, performs immediate syntactical checking, thus considerably reducing a programmer's debugging time.

Syntax checking can be an important function of a program editor. However, many systems having this capability can support only a limited number of languages because the syntax of the language is built into the system. If the syntax checking were table-driven, however, all that would be required to add a new language to the system would be a new syntax table. At this time, it is not clear to the authors whether the time required to perform syntax checking on-line at the editing stage is justified when the user has at his disposal a fast (possibly interactive or incremental) compiler. The next system is also syntax driven, but provides the user with a new technique for constructing his program.

#### Emily, Argonne National Laboratory

With the other editors described in this paper, a user creates his text by typing the appropriate strings of characters. With Emily [10], he creates and modifies program text by selecting among choices presented by the system in accordance with a syntac-

tic description of the programming language. Users interact with the Emily system via an IBM 2250 Graphic Display Unit attached to a 360.

Most programming languages can be described by a syntax, a set of rules specifying one or more replacement strings for each of a set of nonterminal symbols. For example, Figure 3 shows a simple syntax for a small part of PL/I. The nonterminal symbols are those that begin with "<" and end with ">". All nonterminals in the text must be replaced before the program is completed. The rule for a variable <VAR> specifies entry of an identifier from the keyboard or from a display of other identifiers that have already been entered in the program.

Text is created with a sequence of selections. The screen of the 2250 appears divided into three areas: text, menu, and message. The text area in the upper two-thirds of the screen displays the text under construction as a string containing nonterminals. Nonterminals are underlined and one is enclosed in a rectangle to mark it as "current." The menu (the lower third of the screen) displays a set of possible replacements for the current nonterminal. The user selects a replacement rule and the system makes the substitution, locates a new current nonterminal, and displays a new set of choices. The message area is used for entry of identifiers and also displays status and error messages.

Creation of an IF statement might take place in the steps shown in Figure 4. The

```

<STMT> ::= <VAR> = <EXPR>;
        | IF <EXPR> THEN <STMT>
        | DO; <STMT*> END;

<STMT*> ::= <STMT>
        | <STMT> <STMT*>

<EXPR> ::= <EXPR> + <EXPR>
        | <VAR>

<VAR> is an Id

```

FIG. 3. Syntax for small part of PL/I.

- 1) <STMT>
- 2) IF <EXPR> THEN <STMT>
- 3) IF <VAR> THEN <STMT>
- 4) IF FIRST\_TIME THEN <STMT>
- 5) IF FIRST\_TIME THEN DO;  
    <STMT\*>  
END;
- .
- .
- .
- 23) IF FIRST\_TIME THEN DO;  
    FIRST\_TIME = FALSE;  
    SYMBOLS = NULL;  
    END\_TIME = DAYMINUTES + 10;  
END;

FIG. 4. Sequence of steps to generate an IF statement.

initial current nonterminal is <STMT> and the menu would display the three choices shown below.

```
<VAR> = <EXPR>;
IF <EXPR> THEN <STMT>
DO; <STMT*> END;
```

The user would point the lightpen at the second of these, and the system would generate the second line of Figure 4.

The full implemented syntax for PL/I has 24 choices for <STMT>, but excludes most PL/I input/output features. Other implemented syntaxes include GEDANKEN [16], a syntax for outlines, and a syntax for input to a processor to generate syntax tables. The outline syntax has been useful for outlining papers and talks.

A syntax imposes a hierarchical structure on text, and this structure is used to advantage both within the system and in the system facilities available to the user. Each selection from the menu generates a node with space for one pointer for each nonterminal in the replacement string. When a nonterminal is replaced, the corresponding space is filled in with a pointer to the node generated for the replacement. Each nonterminal thus generates a subtree of nodes that

is presented on the display as a string of text.

A user can change his view of the text, so the string generated by any nonterminal is represented by a single symbol called a "holophrast." For example, the IF statement could be displayed with all text generated from the <STMT\*> represented by a holophrast. In larger programs, this feature means that the user can view the structure of his text without viewing the details. Alternatively, he can cause the display to descend into the structure and view the details in full.

Text is also modified in terms of its structure. The text represented by any holophrast can be deleted, moved, or copied. Modification of the file is made easier by allowing a file to contain any number of arbitrarily sized, named text fragments (records) such that text can be moved between them. When text is deleted, it is not destroyed immediately, but is automatically moved to a special system fragment called \*DUMP\*. If a mistake is discovered before the next text modification is made, the deleted text can be retrieved from this dump.

The syntactic formalism developed for the Emily system has a number of interesting aspects. Because the system formats the display of text for the user, the syntax must be specified by the designer with inline formatting codes that specify indentation and new line operations. A special notation for lists allows arbitrary list separators and facilitates user commands to insert and delete list elements. Finally, the identifier block structure can be specified in the notation, and the system offers an interactive cross-reference feature. The user can ask to see all declarations or all instances of any identifier in any given block. As he pushes a button, each instance is displayed in turn.

User engineering was an important consideration in the design of Emily. A certain unity of appearance is achieved because the program text is created, viewed, and modified in terms of its structure rather than as literal (meaningless) character strings. Text expands and contracts from nonterminal to string to holophrast and back. The number of direct user interaction subroutines was

kept small so the user could easily understand the input conventions. For instance, there is only one subroutine that permits the user to type a string of text. Other techniques allow menu picking rather than typing of codes wherever possible. However, Emily's designer reports that preliminary experimentation showed the lightpen to be a poor device for menu selection. The user must aim his whole hand and arm, and he must wait for the system to display the choices. There is nothing to aim the pen at during this wait. A better device might allow a user to select from among twenty or thirty choices with one hand, without having to look at that hand. With such a device, a user could make selections as fast as possible, even without waiting for the display, if he were familiar with the syntax. In dealing with syntax with which he was less proficient, he could wait to make a selection until he has read the choices.

A second problem is that identifier selection is not as easy as selection of syntax rules because identifiers are specific to an individual program. Emily displays existing identifiers for selection, but their location in the menu can change as new identifiers are added.

### FREE FORM TEXT EDITORS

The traditional method of manuscript composition and publishing is a slow, unsatisfactory process. Turnaround time and communications between authors and publishers are so poor that documents are often out of date before they are published. The author's role in the process is perhaps the most old-fashioned of all. His tools include a pencil, a typewriter, some pieces of paper, an eraser, a few reference works, and his mind. The fussy, professional (and probably imperfect) author faces a very real problem in that he has to rewrite his manuscript many times, do research, and refer to previous writings. He cannot store all the information he needs in his own mind. He would like to try different versions and techniques before deciding on the final form; he would like to have his previous

writings and all his references at his fingertips. In other words, he needs the power and memory of a computer to assist him in his creative tasks. Especially to the computer scientist as author, it is esthetically pleasing to utilize the computer in all aspects of his work, including writing and documenting as well as programming.

The most obvious advantage offered by computer-assisted editing is the tremendous reduction in time required to produce a final, or alternate, document. Normal editing requires many cycles in which text must be read and reread, typed and retyped. Text stored in the computer, on the other hand, is never retyped but only updated. As a consequence, the number of typographical errors is reduced and far less proofreading is required. Using an on-line editing system, an author or editor receives cleaner-looking (draft) copy and he receives it more often—when he asks for it, not a day or two later. Thus, an author can afford to be more particular about expressing an idea exactly because it is very inexpensive in terms of both time and effort to print out an intermediate draft. Furthermore, if the system were adapted to handle many users simultaneously, and were available on a full-time basis, the user would never need to request any intermediate printed copy since all his writing and editing could be done on-line.

In summary, then, the advantages of working in this manner include:

- 1) easy access;
- 2) immediacy of response;
- 3) ease of making hard copy without intermediate stages of typesetting, proofreading, resetting, reproofing, etc.;
- 4) reduced "turnaround" time for any type of file research and writing task;
- 5) common access to the same data base (this is useful for a pool of researchers or documenters working in the same area, or for common access to updated (project) management information);
- 6) "constructive plagiarism": the easy modification of previously written materials for present purposes (as in the writing of papers, contracts, proposals, or brochures);
- 7) great simplification of document dissemination.

nation and storage—no hard-copy bulk, but any degree of archival protection desired;

- 8) far greater flexibility for browsing and linking text fragments than with manual methods of working with hard copy; and
- 9) relatively modest cost for all this increase in activity and efficiency, when compared with all aspects of present systems: writing delay, retyping, proof-reading, typesetting, revision of galley and page proof, printing, binding, distribution, storage (and subsequent inaccessibility due to distance, lack of shelf space, poor indexing, borrowed or lost copies, etc.). The user cost includes the machine time used (typically a 5% or smaller rate of CPU utilization/user time) and the rental or purchase cost of the terminals employed.

One of the disadvantages of on-line editing compared to hard-copy editing is the expense; terminals, and especially communications, are still too costly, although prices, particularly of terminals, are coming down rapidly. Also, the convenience and naturalness of hard copy are still important to the “back of an old envelope” school of writing and the “5:07 Stamford express/red pencil” school of editing. Finally, there is the inability to store a record of all the editing changes made (important to magazine publishers, for instance). Because of these disadvantages, the transition from hard-copy to soft-copy techniques will be a gradual one and will require greater cost effectiveness, reliability, and system availability than is obtainable with current hardware and software technology. Also, the considerable problem of economically retaining an “edit trail” record of changes to a text is still only in the research stage [8].

The systems described below are representative of the current state of the art, and range in cost and capability from limited to extremely high. Most of them are not available as a service on the open market, being supplied either as part of a time-sharing operating system for those few lucky enough to have access to one, or existing as a limited access (university) research project. Those discussed first are in many senses less powerful than the sophisticated

time-sharing system program editors. They are discussed in this section on text editors because they are usually marketed as such, and are available on medium-priced computers (without hardware address translation), thereby allowing minimal editing at minimal cost. For example, ATS will run in 64K bytes of an ordinary batch processing 360 and service several dozen terminals. The systems mentioned last (HES/FRESS and NLS) are far more powerful, occupy significantly more core (as much as 100-200K bytes of core residency, and perhaps several times that much for the entire system), and represent many man-years of design and implementation effort. Also, they are not subject to most of the bothersome restrictions imposed by line orientations of the previous systems.

#### **Magnetic Tape Selectric Typewriter (MTST), IBM Corporation, and ASTROTYPE, Information Control Systems**

These two extremely rudimentary “stand-alone” editing systems are similar in their design and intended use, and do not require the services of a large general-purpose computer. MTST [13] consists of a single IBM Selectric Typewriter connected to a small control/memory unit and is used for printing out neat looking (but unjustified) form letters and manuscripts (e.g., legal contracts). The ASTROTYPE system [1] consists of up to four typewriters and memory units connected to one control unit (the DEC PDP-8) and, in addition to preparing forms, is used for preparing input for photo-composition devices. These systems were not designed for massive editing jobs. The only editing allowed in MTST is to substitute equal length character strings; to effect a substitution of a larger for a smaller string, the entire manuscript with the correction must be copied onto a second tape. The correct place for an edit is located by printing out the contents of the tape until the area of the edit is approached; the operator must then manually stop the processor and retype the line.

In both systems, the text is recorded on magnetic tape as it is typed in, and can later be modified and printed in final form. Text can be typed in an “unchangeable”

mode, in which case it is printed exactly as it was typed, or in an "adjustable" mode, in which case the control unit prints blocks of text (e.g., paragraphs) according to the current column width (which can vary).

The editing functions of *ASTROTYPE* are a bit more reasonable than those of *MTST*. Text is stored in lines, which are numbered sequentially from the top of the file. These relative numbers naturally change as new lines are inserted or old lines deleted. The basic editing command is substitute. Substitutions within a line are made by typing the line number, the old character string (plus any additional context that might be required to uniquely identify the text), and the new string (including the additional context used above, if applicable). As in the *CMS* editor, insertions and deletions within a line are degenerate forms of substitution. Thus, to insert within a line, one must specify context to the left or right of the insertion point as the old string and repeat this context plus the text for insertion as the new string; to delete within a line, the text to be deleted must be typed as the old string, and null text specified as the new string. However, there is still the restriction that editing changes must be fairly local because the line number and old and new strings must be typed on the same typewriter line. Individual lines may also be erased and moved by number. Verification is provided by a printout of the line before the change is actually made.

Printing in both systems is done at the typewriter at the rate of 150 words/minute, and various fonts and type sizes may be used by changing the typing ball. The printing can also be programmed to stop in the middle, so that additional input may be entered manually, and then continue. This is especially useful when changing a portion of a form letter.

#### **System/360 Administrative Terminal System, IBM Corporation, and VIPCOM, VIP Systems**

*ATS* [19] and *VIPcom* [24] are very similar commercial editing programs that utilize the *IBM 2741* typewriters as the interactive device. *ATS* is provided by *IBM* as a standard package on the 360 series while *VIPcom*, basically *ATS* with photo-compo-

sition output, was available until recently on a dial-up, service-bureau basis. They are closer to a reasonable editing system than either *MTST* or *ASTROTYPE*, allowing various forms of input and formatted output, and providing a larger set of editing and formatting commands. Except for the photo-composition output of *VIPcom*, they are more primitive than the *CMS* context editor, lacking, for example, pattern-scanning facilities.

As in *ASTROTYPE*, each time a line is typed in, an internal line is created and a line number assigned. The length of an internal line may vary from 0 to 130 characters, and a text file may contain up to 9999 internal lines in a linked-list storage structure similar to that of the *CMS* editor. The line numbers, as in *ASTROTYPE*, are relative to the top of the file and change dynamically as lines are inserted or deleted. This can be confusing because the line numbers of text strings to be edited may no longer correspond to those of the most recent printed copy. The designers of *ATS* suggest that editing be performed from the bottom of the file to the top; this eliminates the problem of changing line numbers, but is not a natural way of working.

Deletions of one or more lines are quite easily made, but to insert new lines in the middle of a file, the lines must first be typed at the end of the file and then moved to the desired position. Text can be moved around by lines, but not copied. Substitutions, deletions, and insertions within a line are made as in *ASTROTYPE*.

Text can be entered in "formatted" mode, in which case the text can be arranged by the output program to satisfy the specified page format (e.g., line justification), or in "unformatted" mode, in which text is saved and printed exactly as it was typed in (useful for rigidly formatted material such as tables). As in the two systems just described, an on-line printout can be stopped in the middle, allowing the user to type in additional text.

Systems such as these are far from ideal for general purpose editing. The terminals used, however, are inexpensive compared to *CRT* displays, and the service required from the *CPU* is minimal because the edit-

ing functions are few and the data structure quite uncomplicated.

#### **A Tablet-Based Editor, Carnegie-Mellon University**

This experimental editor [4] uses hand-drawn proofreader's symbols as the method of editing text displayed on a CRT. The symbols are drawn on a Rand data tablet and are recognized by the program by passing various characteristics of a given symbol through a decision tree (discrimination net). The editing actions recognized include insert, delete, substitute, interchange, move, and scroll. Since this experimental system was not meant for production editing, there are no real paging and file-handling mechanisms, nor are there any hard-copy output facilities.

The editing functions are specified in a very natural manner. As an example of how this type of editing is performed, consider the substitute function. The user draws a line through the text to be deleted. The system deletes this line, blinks the indicated text for verification, separates the text by opening a blank line, and inserts a cursor, enabling new text to be typed in from the keyboard.

The conclusions reached by the experimenters were: 1) that using such a system, vocal input of text could be practical, even with high error rates, because any errors are easily corrected; and 2) that a self-sufficient editing terminal requiring computer intervention only to "fetch, format, and store text" could be constructed, because the decision tree and primitive editing logic could be made part of the terminal's hardware. However, the cost either in CPU utilization for on-line recognition or in hardware development makes this form of editing impractical at present.

#### **The Hypertext Editing System (HES) and File Retrieval and Editing SyStem (FRESS), Brown University**

The Hypertext Editing System [3] is a flexible, CRT-based (IBM 2250) system allowing full editing and formatting capabilities. It is oriented toward "typeset" output (using a computer line printer) as well as flexible input and on-line editing. A light-

pen and a set of "function keys," under program control, are used to indicate to the system the nature of the edit to be performed. The editing function is selected by pressing the appropriately labeled function key. The portion(s) of text to which the function applies are then indicated by pointing at the text with the lightpen. No command codes for the functions need be remembered, and no extra typing is required to indicate a context string.

To delete a portion of the text, the "delete" function key is pressed, after which the two endpoints of the text to be deleted are pointed at with the lightpen. The inclusive text is then blanked out on the display for verification. If the deletion is correct, it is "accepted" by pressing a control key; otherwise it may be "canceled," leaving the original text intact. Among the editing functions are insert, delete, substitute, rearrange, and copy. The system provides prompting messages that specify the actions available at each step. Large strings of text may be edited in one command, although certain implementation restrictions have been imposed. For example, a maximum of approximately 2500 characters may be deleted or rearranged at a time (but major restructuring of text may be done by creating new "branches" between text fragments, as explained below).

In addition, many formatting options are available so that text may be formatted both for on-line display and hard-copy printouts. An off-line computer typesetting program, IBM's TEXT 360 program [21], is used for final hard-copy printing on a computer line printer equipped with an upper and lower case print chain.

The data structure, and hence the editing operations are entirely independent of display or printout lines and pages. Text is externally segmented by the user into arbitrarily long user-designated fragments called "text areas" (see Figure 5). Each area is a continuous linear string of text, like an Egyptian scroll, and might be a chapter, an entire book, or a short footnote. These areas may be interlinked and cross-referenced in any manner so as to form a directed graph of text segments (the vertices of the graph) and their cross references (the edges). Two

types of cross references exist: "branches" are unconditional jumps between two fragments that the user may encounter in the text (typically in a "menu") and which force him to lightpen a choice in order to proceed, while "links" are conditional jumps, which the reader may bypass or lightpen. The link, in effect, is an on-line generalization of the manuscript footnote principle that invites digressions, ancillary explanations, and browsing.

The resultant mobile of text fragments is called a hypertext, "the combination of natural language text with the computer's capacities for interactive, branching, or dynamic display ... a nonlinear text ... which cannot be printed conveniently ... on a conventional page ..." [14]. A practical example of a hypertext might be an on-line encyclopedia or a set of programming and systems reference manuals, with each cross-reference lightpen sensitive. Functions are provided that allow the fragments of text to be interpreted and examined in a variety of ways, in particular to have linear paths traced through the hypertext, either for on-line browsing purposes or for printing in a conventional manuscript form. The system also remembers the sequences of branches or links that the reader has taken, and this allows him to reverse his trail. Random access to any point in the text is provided by allowing the author to assign "labels" anywhere in the text and later to "jump" to any of them by lightpenning the appropriate one from an alphabetized menu.

A given area of a hypertext is stored internally as one or more pairs of variable-sized "pages" (unrelated to a printed page), each pair consisting of an "order code" page and a "text" page. The order code page is an ordered sequence of text pointers (displacements into the text page) interspersed with formatting and structure codes, while the text page contains the actual literal strings of characters. To accomplish an edit, order codes and their pointers are inserted, deleted, or updated while the text strings remain intact. An edit therefore involves little character shuffling, at the expense of rapidly accumulating "garbage" strings.

Although HES is in production use in a number of 2250 installations for the writ-

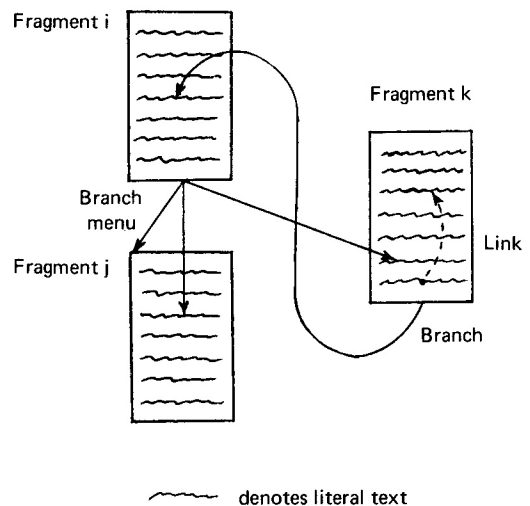


FIG. 5. Model of a simple Hypertext.

ing of manuals, theses, and proposals, it was essentially an experimental system for studying text handling techniques as well as applications of hypertext as a new medium. FRESS [23] is a commercially available production version of the system (for the 360/67 as well as the standard 360) supporting a spectrum of terminals in such a way that all functions are available even on the lowest power terminal.

The standard terminal is a 2741 typewriter, best used for initial input and minor editing of small files; it is also used as an acoustically coupled remote terminal and for remote hard copy. Also supported is the IBM 2260 CRT alphanumeric display (and its equivalents), used for faster and more convenient editing and on-line browsing. More powerful CRTs operate as in HES, and are intended for major restructuring and rapid editing (rush jobs). New types of terminals can be supported by simply including a new program module to handle device-dependent input and output, and to convert editing requests into their standard internal representation.

Facilities not in HES but implemented in FRESS include completely arbitrary size string edits, pattern scanning, keyword retrieval, photo-composition output, interfile linking and editing, and protection of files and blocks of text by passwords that can have a mask enabling or disabling each of



the hundred-odd commands individually. Section outline facilities, similar to those described in the section on NLS, are currently being debugged. As an example of string editing on a 2741, a substitution ("change") of the string "New sentence" for a large sentence that starts with the string "The text..." and ends with "end" could be specified as

c/The t . . . end/New sentence

where the "/" again denotes an arbitrary delimiter and the ellipsis is a pattern that matches all characters between the initial "t" of "text" and the first occurrence of "end".

Unlike the separate text page/order code page data structure of HES, the data structure for FRESS is a set of arbitrarily long, continuous (but naturally internally paged) strings of text with "order codes" (which indicate formatting and structure) directly embedded within the text. Editing is done by operating directly on the character strings, thereby causing considerable moving of characters (spreading them apart for an insert or closing them up for a delete) but avoiding the rapid buildup of garbage strings that occurs in HES. Additionally, tables of pointers are used to identify important internal places in the strings, such as those that "anchor" labels or cross-referencing links.

#### **NLS, Augmented Human Intellect Research Center, Stanford Research Institute**

The work being done by the AHI group at Stanford University is most impressive. Their system (NLS) embodies much more than just a text editor; their aim is to provide a new way of thinking and working by utilizing the power of the computer in all aspects of one's work [9].

We are concentrating fully upon reaching the point where we can do all of our work on line—placing in computer store all of our specifications, plans, designs, programs, documentation, reports, memos, bibliography and reference notes, etc., and doing all of our scratch work, planning, designing, debugging, etc., and a good deal of our

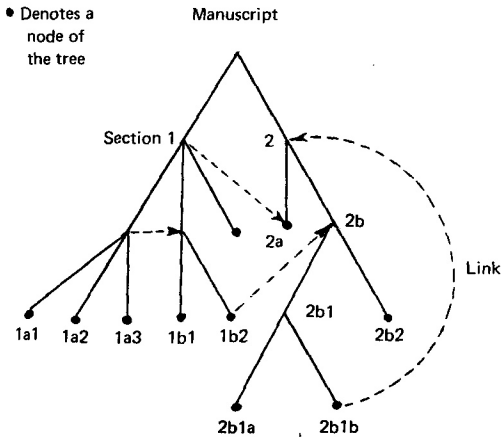
intercommunication, via the consoles [9].

In this survey it is not possible to describe adequately the many features of this powerful system; the AHI paper [9] is highly recommended. Currently, the Center has a time-shared SDS 940 (65K, 24 bit core) with a 4.5 megabyte swapping drum and a 96 megabyte file storage disk. At the time this paper was written, the facility was being upgraded to a PDP-10 configuration, in conjunction with the NLS center becoming the information utility for the ARPA network [18], handling all its documentation, archival storage, and information dissemination. The 940 system, while capable of handling teletypes, is primarily dedicated to twelve "work stations." These are standard television monitors, displaying the output of small CRTs, and equipped with a "mouse" for pointing to text and a one-hand five-key handset for specifying commands to the system. The handset can also be used to input text, although a standard keyboard is provided.

Of particular significance is the ability of NLS to display a file from many different points of view. For example, the hierarchical outline structure of a text—the various headings—may be stored as part of the data structure, and one may ask to see, for example, only section 3.1.4, or all sections down to two levels of subsectioning, or the first line in each of the subsections on the fourth level. The text is thus viewed as a collection of sections (called "statements") with a tree structure superimposed on this basic data structure. Each statement (less than 3000 characters) is meant to contain a single complete thought or idea, but may have substatements down to an arbitrary number of levels (see Figure 6). Most standard tree manipulations are allowed at a given level in the tree; e.g., locating or deleting the next node or the previous one, locating the first subnode, rearranging neighboring nodes, etc. Note that this hierarchical approach to files, in contrast to the continuous string approach of HES and FRESS, is useful for documents as well as programs.

Many ways of viewing the text exist (de-





Each node is a text section or statement. The dotted connections represent optional cross references and associations between statements (links).

FIG. 6. Hierarchical statement structure of NLS.

terminated by "viewspecs," e.g., displaying only the first "n" levels of the tree or the first "n" characters of each statement) and elaborate methods for jumping around in the text are available (by complex pattern searches, by the statement outline structure, keyword retrieval, etc.). Cross references, similar to HES/FRESS links, may be made between statements in the hierarchy.

The editing commands are extensive and specialized, e.g., insert character, insert word, and insert statement are all separate commands. Most editing is done on a statement basis so that one cannot, in one operation, substitute for a string that crosses statement boundaries. One can, however, move part of a statement into another. Editing within a statement is done by copying the updated literal character string to an available place in an internal page. Since the text is hierarchically arranged, rearrangements of the tree structure on a statement basis (causing automatic renumbering) are easily and economically made; the statement hierarchy is encoded by explicit pointers to the pages on which the individual statements are stored, so that only pointers, not text, need be moved.

Statements may contain such embedded control codes as font changes, keywords

(attributes), user names (labels), cross references to other statements, etc.; these codes are contained inside syntactic delimiters (e.g., brackets) within the statement text. They can therefore be treated as ordinary editable text until their special purpose is invoked by a command, and this allows the knowledgeable user to modify orders using ordinary text-editing commands rather than having to invoke specialized "change order" commands. PASS4 in NLS provides hard-copy formatted output for a variety of devices, such as the line printer.

### EDITING FEATURES OF TYPICAL TERMINALS

The most common editing terminal today is still the teletypewriter (Teletype model ASR33 or IBM 2741 and their equivalents), principally because of their relatively modest cost—\$800 for the simplest teletype to approximately \$3,500 for the 2741. In particular, acoustically coupled teletypewriters running up to 300 baud (15–30 characters/second) provide a convenient, portable terminal for original input, program editing, and "minor surgery" on ordinary manuscripts. For free-form, general-purpose text editing, however, the teletypewriter's advantages of low cost and hard copy transcripts are outweighed by the disadvantage of slow and noisy "display." An alphanumeric CRT display, capable of rapid display of many hundreds of characters and local editing facilities is almost essential.

The price of such alphanumeric terminals has decreased steadily over the last several years, while the available capabilities, in terms of number of displayable characters and editing options, have increased. A continuum of editing power is available in these terminals, starting from a small CRT (selling for as little as \$2,000) to replace the hard-copy teletype and ranging to a \$15,000 large-screen terminal that is capable of local interactive text editing. Below are several examples intended to give the reader an idea of the cost/power combinations that are available along the spectrum. (The teletype replacements have not been

considered, because they have no editing capabilities to speak of.)

The IBM 2260/1 is a simple buffered CRT capable of displaying 960 characters (12 lines with 80 characters each) from a 64 symbol character set. Cursor manipulation using up/down/left/right/newline controls is supplied to assist in the two functions available, which are character input and character overstrike (the only real editing capability). In single CRT quantities, the 2260 is prohibitively expensive, as it requires a buffer/display controller—the IBM 2848/3. However, since one 2848/3 can control up to eight 2260/1 CRTs, the equipment, if used in quantity, can be cost effective, renting at approximately \$150/terminal/month.

The Computer-Optics CO:70 is a more sophisticated device for text editing than the 2260. Basic cursor controls of up/down/left/right are enhanced by the repeat function, which facilitates rapid cursor positioning. Hardware editing capabilities—character and line insert and delete and character overstrike—are available in a system that sells for approximately \$8,500. The terminal may be used stand-alone to edit text stored on magnetic tape, or may access data bases on a mainframe computer over low-speed communication lines. Additionally, the ability to display 3000 characters from an 88 symbol ASCII character set, which includes upper and lower case, make the CO:70 more useful as a text, rather than program, editing terminal.

The Computek model 100 is a microprogrammable terminal capable of displaying 1000 upper and lower case characters, with the potential of doing nontrivial editing locally. It can be specially microprogrammed to edit a database stored on a cassette or other local memory, without using the resources of a mainframe computer. The basic firmware supplied with the system provides character, word, and line insert and delete for approximately \$4,500, with possibilities for expanding Read Only Memory to handle more complex functions. In the Computek, high speed is sacrificed in favor of low cost. Use of relatively slow, but inexpensive delay-line memory causes reduced editing

speed as the complexity of the required functions increases. For example, \$5,000 is sufficient to purchase a Model 100 capable of performing a rearrange in addition to the standard functions listed above, but the response time for a simple rearrange may be on the order of seconds! Because of the local editing ability, useful work can be accomplished with the terminal attached to a general-purpose computer over low-capacity telephone lines.

The Harris 1100 is similar to but more powerful (and expensive) than the Computek 100. The basic system consists of a 2000 character CRT (with single and double column formats), 2K bytes of MOS shift register memory, a 100-symbol character set, and keyboard and function keys; it sells for \$14,500. Display buffer memory is expandable in 2K byte increments, to a maximum of 6K bytes, at a cost of \$2,000 per increment to provide buffering of and scrolling through a 6000-character data base. Editing may be accomplished on-line via telephone communications lines with capacities up to 1200 characters per second, or off-line, using paper tape for all I/O. Standard features include a cursor with up/down/left/right/newline/home controls (and, unfortunately, no repeat key), a scroll function that is designed for systems with more than 2K bytes of memory, and a nice range of editing capabilities. In addition to the standard character editing functions of insert, delete, and overstrike, these include "block" editing commands: by inserting special "define block" symbols in front of and immediately following an arbitrary string of text, a block may be defined and then edited (either deleted or rearranged—moved to the current cursor position) or output (either to paper tape or a computer, depending on the mode of operation). The terminal is oriented towards copy editing, and is used by a number of newspapers and printers in the United States and Europe.

These types of terminals, with no local writable memory other than the display buffer, are excellent tools for editing simple linear text, but lose much of their usefulness when being used on a data base that is not pure text or is not stored in screen image

format. For example, if the text contains typesetting codes such as skip lines, indent, etc., whose effect is to be shown on the screen, a translation must be made between the internal data structure and the external appearance of the text, and vice versa. More seriously, checks must be made to prevent the editing of nonexistent blanks (those skipped by a paragraph, for example) or typesetting codes that have global scope in the text (e.g., indent). If, therefore, the computer cannot just place the updated text back into the data base, then it must be apprised of the changes made on the screen by the terminal (to check them for validity and then to execute them). The terminal must either transmit the changes made to the displayed text operation by operation, or the program in the mainframe must let local editing take place for a while, and then interrogate the resultant local buffer. The mainframe program must then compare (display buffer to display buffer) what was sent to the terminal with what came back, decompiling the commands from the modified display buffer to check and modify the data base. The first process is expensive in input/output and the second in CPU usage.

An answer to the problem of handling structured or formatted text could be to transmit a section of the structured data base to a remote terminal and have it locally modify the data base with its own (primitive) data-structure editing routines, transmitting the updated data base back to the mainframe on a batch basis. To accomplish this task, local computation and read/write storage are required in addition to the display buffer.

The IMLAC PDS-1, an example of a device capable of local data-structure editing, is a small computer with up to 32K 16 bit words of local memory. It is integrated with a CRT that uses the PDS-1 core for a display buffer. It has no character generator, and characters are displayed through the use of character-stroke subroutines, executed by the scope. Cursor manipulation is handled by software in the PDS-1. Rudimentary vector graphics are also provided, and the total package (with 4K bytes of

core and a lightpen) can be obtained for approximately \$10,000.

## CONCLUSION

On-line composition and editing of programs, coupled with interactive debugging, has become an established, cost-effective use of computers. Similarly, minor text editing, such as the correction of typographical errors in memoranda, is cost effective, since only teletypewriter consoles and minimal service from the CPU are required. In contrast, the imaginative use of computers for on-line composition and extensive manipulation of free-form text is still in the early stages of experimentation and user conversion. This is due partially to the high cost of CRT terminals that provide the human factors essential to general purpose editing, partially to the high cost of system resources and implementation time for the sophisticated programs required, and partially to the long time required to wean users from traditional off-line hard-copy processes.

Hardware prices are coming down steadily, however, and as more users switch to on-line thinking, creating, and manipulating, this use of computers will become increasingly accepted (and judged cost effective), hopefully to the same extent that numerical and data processing applications are already considered to be a legitimate use of the computer.

## POSTSCRIPT

As a matter of interest, it may be noted that the authors wrote and edited this paper using several text-editing systems. The paper was originally composed on-line by "scissoring" and modifying co-author Rice's Master's Thesis and several other documents, using the lightpen of the 2250 display console of the Hypertext Editing System. The hard copy of this first draft was then reviewed by a number of workers in the text-editing field. Subsequently the file was converted to a FRESS file when HES was phased out, and then edited some half dozen

times on the 2741 typewriter (due to economy measures, the 2250 and 2260s had "recessed" in the meantime).

The final manuscript was reviewed by Wilfred J. Hansen, of Argonne National Laboratories. In order to save time and to meet the publication deadline, his rewrites of the TVEDIT and Emily sections were typed into Argonne's text-editing system and printed out on a dial-up 2741 at Brown University. Compromise wording was arrived at by telephone the same day. Unfortunately, such long distance cooperation with computer intermediary is still primitive; for example, due to file incompatibility the rewrite had to be re-keystroked into FRESS, Brown's text editor. Furthermore, the entire paper had to be re-keystroked in order to be phototypeset. In the future, we may hope that such duplication of effort will disappear, and that files can be cooperatively worked on in real-time, regardless of geographic location or editing system, and still at reasonable costs.

## REFERENCES

1. "ASTROTYPE." Form No. 30, Automatic Office Division, Information Control Systems, Inc., Ann Arbor, Mich.
2. BRATMAN, HARVEY; HIRAM G. MARTIN; AND ELLEN C. PERSTEIN. "Program composition and editing with an online display." *Proc. AFIPS 1968 FJCC*, Vol. 33, Pt. 2, AFIPS Press, Montvale, N.J., pp. 1349-1360.
3. CARMODY, STEVEN; WALTER GROSS; THEODOR H. NELSON; DAVID RICE; AND ANDRIES VAN DAM. "A hypertext editing system for the /360." In *Pertinent concepts in computer graphics*, M. Faiman and J. Nievergelt (Eds.), Univ. Illinois, Urbana, Ill., March 1969, pp. 291-330.
4. COLEMAN, MICHAEL K. "Text editing on a graphic display device using hand-drawn proof-reader's symbols." *Ibid.*, pp. 282-290.
5. "A conversational context-directed editor." IBM Cambridge Scientific Center Report, Form No. 320-2041, Cambridge, Mass., March 1969.
6. DENNING, PETER J. "Virtual memory." *Computing Surveys* **2**, 3 (Sept. 1970), 153-189.
7. DEUTSCH, L. PETER; AND BUTLER W. LAMPSON. "An online editor." *Comm. ACM* **10**, 12 (Dec. 1967), 793-799.
8. ELLIOTT, W. DAVID; WARREN A. POTAS; AND ANDRIES VAN DAM. "Computer assisted tracing of text evolution." *Proc. 1971 FJCC*, Vol. 37. (To be published.)
9. ENGELBART, DOUGLAS C.; AND WILLIAM K. ENGLISH. "A research center for augmenting human intellect." *Proc 1968 FJCC*, Vol. 33, Pt. 1, AFIPS Press, Montvale, N.J., pp. 395-410.
10. HANSEN, WILFRED J. "Creation of hierarchic text with a computer display." Argonne National Laboratory Report ANL7818, July 1971.
11. LESSING, LAWRENCE. "The printed word goes electronic." *Fortune* (Sept. 1969), 116-119, 188-190.
12. MCCARTHY, JOHN; DOW BRIAN; GARY FELDMAN; AND JOHN ALLEN. "THOR—a display based time sharing system." *Proc. AFIPS 1967 SJCC*, Vol. 30, AFIPS Press, Montvale, N.J., pp. 623-633.
13. "Magnetic tape Selectric typewriter." Forms No. 543-0510-1, 543-0515, 549-0204, and 549-0700, IBM Corp., New York.
14. NELSON, THEODOR H. "Getting it out of our system." In *Information retrieval: critical view*, George Schecter (Ed.), Thompson Books, Washington, D.C., 1967.
15. *QED reference manual*. Com-Share, Ann Arbor, Mich., Ref. No. 9004-4, Jan. 1967.
16. REYNOLDS, JOHN C. "GEDANKEN—a simple typeless language based on the principle of completeness and the reference concept." *Comm. ACM* **13**, 5 (May, 1970), 308-319.
17. RICE, DAVID E.; AND ANDRIES VAN DAM. "An introduction to information structures and paging considerations for online text editing systems." To be published in *Advances in information systems science*, J. Tou (Ed.), Vol. 4, Plenum Press, New York, 1971.
18. ROBERTS, LAWRENCE G.; AND BARRY D. WESSLER. "Computer network development to achieve resource sharing." *Proc. AFIPS 1970 SJCC*, Vol. 36, AFIPS Press, Montvale, N.J., pp. 543-549.
19. "System/360 administrative terminal system." IBM Corp. Application Program, Manuals: *Application description manual*, No. H20-0297-2; *Terminal operations manual*, No. H20-0589-0; *Program description manual*, No. H20-0582; and *Console operations manual*, No. H20-0590-0.
20. *TECO, Text Editor and Corrector reference manual*. Interactive Sciences Corp., Braintree, Mass., June 1969.
21. *TEXT360 reference manual and operating guide*. Program No. 360D-29.4.001, IBM Corp., New York, 1968.
22. TOLLIVAR, BRIAN. "TVEDIT." In Appendix I, *Thor reference manual*, Stanford Computation Center, Stanford, Calif., Aug. 1966, pp. 104-107.
23. VAN DAM, ANDRIES. *FRESS (File Retrieval and Editing System) user's guide*. Text Systems, Inc., Barrington, R.I., July 1971.
24. *VIPcom user's guide*. VIP Systems, Washington, D.C., Aug. 1969.
25. WALTER, GERARD O. "Typesetting." *Scientific American* **220**, 5 (May 1969), 60-69.
26. *WYLBUR Reference Manual* (Revised 3rd Ed.). Stanford University, Stanford, Calif., March 1970.